ROBUST INTERPOLATION AND ITERATIVE LEARNING STRATEGIES FOR MODULATED TOOL PATH PLANNING

by

Christopher DiMarco

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Mechanical Engineering

Charlotte

2016

Approved by:

Dr. Christopher Vermillion

Dr. John Ziegert

Dr. Scott Kelly

ABSTRACT

CHRISTOPHER DIMARCO. Robust interpolation and iterative learning strategies for modulated tool path planning.
(Under the direction of DR. CHRISTOPHER VERMILLION and DR. JOHN ZIEGERT)

During single-point metal turning processes, a long, razor-sharp chip nest is often formed that can be a hazard to both operators and the part being machined. To combat this, a process called Modulated Tool Path (MTP) machining was developed by Barkman et al [1] that superimposes a sinusoidal motion on the feed direction of the tool path, causing the tool to remove itself from the cut and break chips. As determined by Berglind [2], the amplitude and frequency characteristics of this sinusoidal motion influence the material removal rate, chip length, and surface finish of the part, and thus, must be replicated by the machine tool controller as faithfully as possible. After the appropriate parameters for each MTP movement segment have been selected, the machine tool controller must attempt to reproduce these movements as closely as possible. In order to achieve this, a typical controller will rely on two components:

1. An interpolation strategy, responsible for enforcing physical limitations of the tool, such as acceleration and jerk limits, and

2. A closed-loop controller to ensure that the machine tool is reaching towards the setpoint determined by the interpolator at each loop closure.

In this thesis, an alternative strategy for item 1 is proposed, as well as an iterative-domain technique to improve closed-loop control of repetitive processes. In order to tailor the standard interpolation strategy (known as jerk-limited linear interpolation) for MTP manufacturing, a technique called Exponential and Sigmoidal (E/S) interpolation has been developed which shows a marked improvement in both tracking

error and acceleration required for a given trajectory. While item 2, the closed-loop controller (which is typically a PID type), has been researched in tremendous depth, it has no provisions for improving its performance during repeated tasks or for rejecting iteration-varying disturbances. To combat the first deficiency, an iterative-domain controller such as Proportional-Derivative Iterative Learning (PD-ILC) could be applied, however, the second criticism remains. This thesis proposes a variant of PD-ILC called Disturbance & Performance-Weighted ILC (DPW-ILC) which weights the relevancy of prior control inputs and errors based on criteria of merit, and determines an appropriate control input for the current iteration. As one of the main criteria for DPW-ILC is the measure of the disturbance, it is robust to iteration-varying disturbances. This thesis will show the vast improvement of DPW-ILC versus PID control only and a version of iterative control called PD-ILC in the face of these types of disturbances.

DEDICATION

I dedicate my thesis work first to my family. Mom, Dad, and Tim, you have my deepest gratitude for your endless love and support.

I also dedicate this thesis to my dearest friends for keeping me sane through the challenges of my college experience. Henry, Brandon, and Walter, you have done more for me than you could possibly know.

ACKNOWLEDGEMENTS

I would like to begin by thanking my advisors, Drs. Christopher Vermillion and John Ziegert, for their unwavering support, wealth of knowledge and experience, and profound guidance. Thanks to Dr. Vermillion, Dr. Ziegert, and Dr. Scott Kelly for being on my thesis committee.

I would like to thank Luke Berglind, Christian Lomascolo, Bill Barkman, and Tom Murray for their insight regarding MTP and the development of the Sigmoidal interpolator.

I would like to thank the members of Dr. Vermillion's lab group, including Nihar Deodhar, Alireza Bafandah, Ali Khayat Baheri, Parvin Nikpoorparizi, Shamir Bin-Karim, Mitchell Cobb, Joe Deese, and Brian Smith for their help in working through the creation and clarification of DPW-ILC.

Lastly, I would like to thank the Mechanical Engineering Department, College of Engineering and Mosaic Computing for providing me with access to a fantastic staff of professors with phenomenal experience, a clean and enjoyable working atmosphere, and access to computing facilities.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION TO MTP

During most metal machining procedures, as a tool tip cuts through the metal part, a "chip" of continuous removed material is formed (Figure 1.1). In the specific process of turning on a lathe, where the workpiece is rotating and the tool tip is slow-moving, the chip that is formed can often approach meters of length and coil around the part. This coil is usually very sharp metal, and at high rpms can damage the part's surface finish or even be hazardous to the machine tool's operator.

Figure 1.1: Chip nest created by a standard metal turning process. These are often dangerously sharp and can damage the part and surface finish.

There are several techniques designed to combat this formation of lengthy chips. The "low-tech" solution is simply to have a technician with gloves periodically stop the machine and break the chip. The more commonplace technique employs a chip-breaking tool tip that forces the metal to curl up and occasionally break.

A third solution, created by William Barkman et al [1], uses a standard tool tip, but oscillates it in and out of the cut in order to break the chip. This Modulated Tool Path (MTP) procedure [2] superimposes a sinusoidal oscillation tangent to the feed direction of the tool tip as in Figure 1.2.

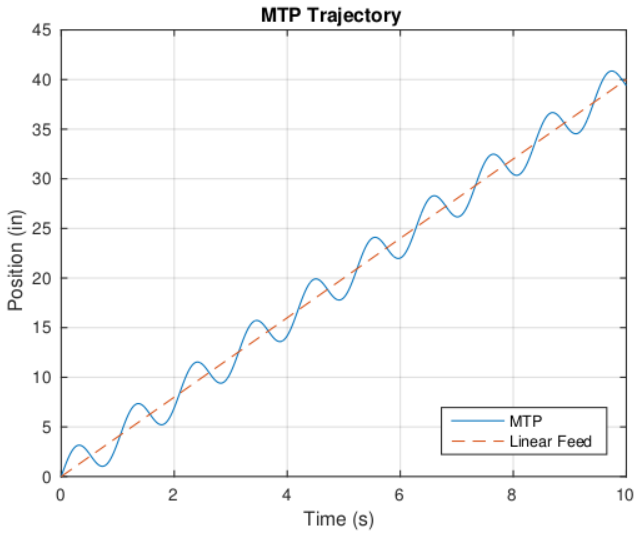

Figure 1.2: MTP Trajectory in a single axis move.

This "ramped sinusoid" is not limited to linear tool movement; the oscillation can be superimposed on any curve or polyline, but we will constrain ourselves to linear feeds only in this thesis. The nature of the oscillation is highly critical, as both the amplitude and frequency can have a profound impact on the length of the chip created by each oscillation, and the surface finish of the final part [2]. As such, it is critical that a machine tool controller be capable of replicating the reference MTP trajectory as faithfully as possible.

Generally, the MTP reference trajectory is programmed into an NC part program as a 6-point sine wave (see Figure 1.3), or a series of linear segments which provide a length, direction, and desired feed speed for the tool to move. The desired feed and position are then interpreted by the controller's interpolation algorithm in order to send the machine tool's servo-control system a position set point for each closure of the control loop. The jerk-limited linear interpolator as outlined by Altintas [4] is a common choice, as it is designed to maximize fidelity to the commanded path while respecting critical jerk and acceleration limits for the machine tool. Unfortunately, due to the design of the jerk-limited linear interpolator, oscillatory processes such as MTP can suffer large falloffs in both amplitude and frequency generation [3].

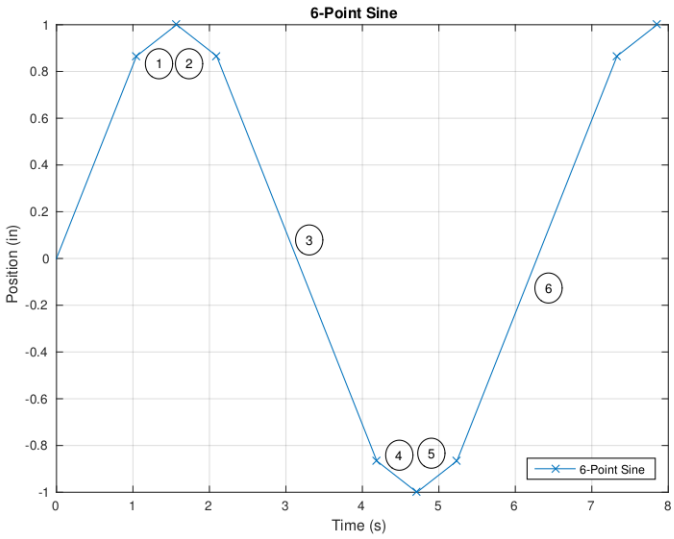

Figure 1.3: MTP 6 Point Sine Wave Oscillation.

After the interpolator has generated the setpoints for the servo-control system, the closed-loop control is tasked with enforcing those targets. Time-domain controller research is plentiful: tracking and contour error improvements in servo control [8] & [10], feedforward and cross-coupling control techniques [7], shaping of reference trajectories [9], and applying multi-objective optimization to control [11]. Unfortunately, these strategies rarely make use of any learning during repeated processes and correcting errors over time, *even though servo error is often a stored dataset*. Following from this, these control systems also have no method for minimizing transient errors, or errors stemming from disturbances that vary in the iteration domain. One such example of this is part expansion due to temperature fluctuation. Naturally, these errors can lead to significant degradation of performance in oscillatory processes like MTP. Some techniques have been proposed to deal with these situations, such as a neural-networking procedure [19], but these require operator input to determine successful passes, and a more internalized process should be developed.

It is the intention of this thesis to propose two improvements for MTP machining:

1. Develop an interpolation algorithm that is more aptly suited to oscillatory trajectories, and

2. Develop a learning algorithm that can:

   (a) Improve accuracy over the span of many iterations

   (b) Continue that improvement in spite of iteration-varying disturbances

1.1 Organization of Thesis

The remainder of this thesis is presented thusly:

Chapter 2 documents the issues with jerk-limited interpolation, and presents the Sigmoid and Exponential Interpolator.

Chapter 3 presents the current control methods of time-domain PD control, and iterative-domain control using PD-ILC, as well as their failings for MTP and iteration-varying disturbances.

Chapter 4 presents the framework of DPW-ILC, as well as two applications of the strategy.

Chapter 5 concludes the thesis and presents attractive options for future work.

CHAPTER 2: INTERPOLATION TECHNIQUES

After a reference trajectory is decided and entered into a machine tool (usually via a NC part program), it is up to the machine tool's controller to plan out the trajectory that will be followed in order to enforce acceleration and jerk limitations on the tool tip in order to prevent damage to the machine tool. Traditionally, this is done using a method called *linear interpolation*.

A simplistic implementation of linear interpolation is laid out by Altintas in [4]. There are three basic segments in the method: acceleration to achieve the desired feed speed, running at the desired feed speed, and accelerating to the speed of the next segment. In order to achieve these three segments, jerk is limited to only three options: maximum positive jerk, zero jerk, maximum negative jerk. After integration, this gives rise to the characteristic trapezoidal acceleration curve, as shown in Figure 2.1. Integrating further, the final position of the tool tip is shown. The method of determining the length of each of these segments is mathematically simple, but real implementation depends on a variety of nested if/then statements to determine the appropriate switching times. It can, therefore, be considered computationally complex, especially for a process such as MTP, which consists of many small movements.

A highly relevant second artifact of this methodology can be seen in Figure 2.2. As the desired trajectory is a sawtooth-type oscillation, the linear interpolator smooths out the oscillations and also includes a varying delay due to startup and feedrate changes. In normal machining, this is rarely a cause for concern, but in MTP machining this can lead to inaccuracies in amplitude or frequency depending on the settings of the machine tool. If the machine tool is configured to complete every move, this

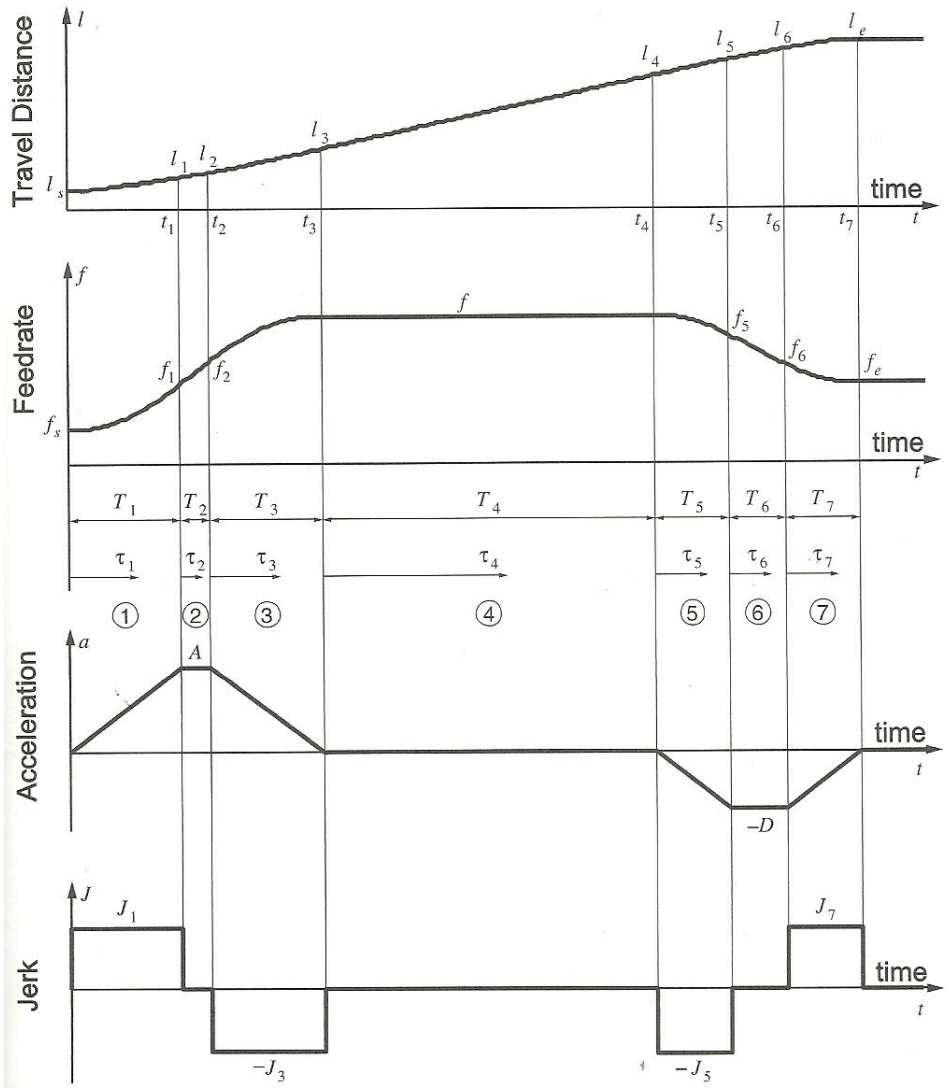

Figure 2.1: Profile of Altintas's Jerk-Limited Linear Interpolation Strategy [4].

will cause frequency falloff, as it will achieve the desired amplitude but will be unable to match the desired frequency. If, on the other hand, the machine tool is configured to start the next move at the start of the next move target start time, this will result in frequency matching with amplitude falloff. These results have been observed by Wes Love and can be found in [2]. While improvements in machine tool tracking have been investigated in the fields of advanced controller design [7], improving contour accuracy [8] [10], input shaping [9], and high performance [11], these enhancements all make use of the standard linear interpolator and are subject to its pitfalls.

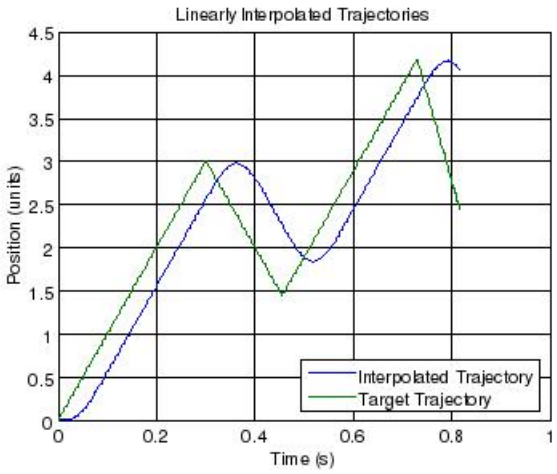

Figure 2.2: Linear Interpolation Result of a Ramped Sawtooth Oscillation.

2.1 Exponential and Sigmoid Interpolators

Given the deficiencies of jerk-limited linear interpolation as applied to MTP machining, an alternative strategy, called the Exponential and Sigmoidal Interpolator, is proposed. The exponential interpolator is based on work by Rymansaib [5] and demonstrates the usefulness of this technique to sinusoidal trajectories. The exponential interpolator is then augmented by the sigmoid interpolator, which is a strategy for joining two segments as well as ending a moment at zero velocity.

2.1.1 Exponential Interpolator at Startup

For Modulated Tool Path machining, each movement can make use of a linear component for feed and a sinusoidal component for chip breaking. Making use of superposition, these two components can be evaluated separately and then summed to get position, velocity, acceleration, and jerk values. Using the equations that define these values, acceleration and jerk limits can be imposed via an convergence-aggressiveness coefficient.

2.1.1.1 Linear Component

The linear component of the exponential interpolator is detailed in Rymansaib [5] and described here. The linear equation in 2.1 is the baseline desired position (x) and

slope (m) at time t:

$$x = mt + b \tag{2.1}$$

For initial startup, the exponential equation for the sigmoid interpolator is:

$$S_L = 1 - e^{-\alpha t^3} \tag{2.2}$$

where α is a constant that dictates the quickness of convergence. The order of the time variable t is 3 to maintain a zero value at $t = 0$ for acceleration, as will be shown below. This leads to a sigmoid interpolated position value x_L of:

$$x_L = xS_L = (mt + b)(1 - e^{-\alpha t^3}) \tag{2.3}$$

It is important to note that at $t = 0$ and for sufficiently large values of t, $x_L = x$. The velocity, $\dot{x}_L$, is evaluated as:

$$\dot{x}_L = m - e^{-\alpha t^3}(m - 3\alpha m t^3 - 3\alpha b t^2) \tag{2.4}$$

which again is equal to 0 at $t = 0$ since all t-terms = 0 and exponential terms = 1, leading to $m - m = 0$. Differentiating a second time, the acceleration, $\ddot{x}_L$, is equal to:

$$\ddot{x}_L = e^{-\alpha t^3}(-9\alpha^2 b t^4 - 9\alpha^2 m t^5 + 6\alpha b t + 12\alpha m t^2) \tag{2.5}$$

At $t = 0$ acceleration is then equal to zero. With $\ddot{x}_L$ set equal to a physically realizable acceleration limit and the values of t known for the movement, a maximum allowable value for α can be determined numerically.

The result of this startup exponential is shown in Figure 2.3 for several values of α. Note that as α increases, the planned trajectory reaches the target path more quickly, and that the tool will exceed the desired feed rate for a small period of time.

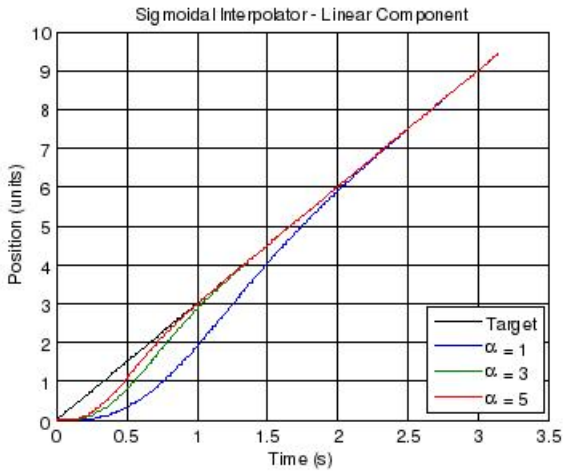

Figure 2.3: Exponential Startup for a Linear Move. As α increases from 1 to 5, the exponential function more closely approximates the targeted linear move.

2.1.1.2 Sinusoidal Component

The desired sinusoidal component is given by:

$$x = A \sin \omega t \tag{2.6}$$

where A is the sine wave amplitude and ω is the angular frequency. As with the linear component, the sinusoidal component is multiplied by an exponential to control velocity and acceleration. The exponential form is similar:

$$S_S = 1 - e^{-\alpha t^2} \tag{2.7}$$

However, the order of t is decreased to 2 as it is not required to be increased to maintain zero acceleration at time step zero in this case. The position equation for the sinusoidal component is generated by multiplying these together:

$$x_S = xS_S = (A \sin \omega t)(1 - e^{-\alpha t^2}) \tag{2.8}$$

Differentiating twice yields the velocity and acceleration equations:

$$\dot{x}_S = 2A\alpha t e^{-\alpha t^2} \sin \omega t - A\omega e^{-\alpha t^2} \cos \omega t + A\omega \cos \omega t \tag{2.9}$$

$$\ddot{x}_S = -Ae^{-\alpha t^2}(4\alpha^2 t^2 \sin \omega t + \omega^2 e^{\alpha t^2} \sin \omega t - 2\alpha \sin \omega t -$$

$$-4\alpha\omega t \cos \omega t - \omega^2 \sin \omega t) \tag{2.10}$$

At $t = 0$, both velocity and acceleration equal zero. As with the linear component, since all values other than α are known, the maximum value can be computed with a numeric solver to ensure acceleration and velocity maximums are not exceeded. Figure 2.4 is an example of the interpolated trajectory for several values of α. Figure 2.5 shows the interpolated trajectory when both a linear component and sinusoidal component are summed.

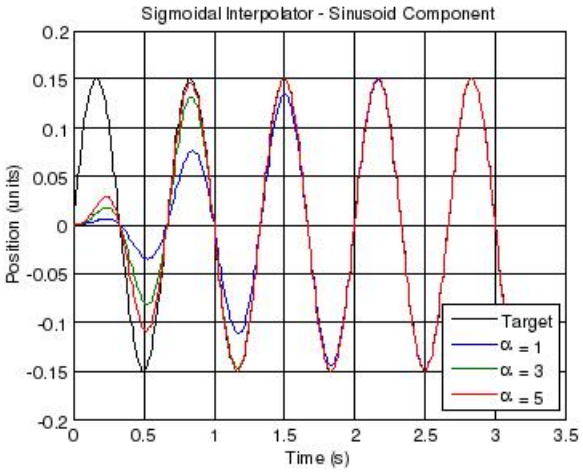

Figure 2.4: The sinusoidal exponential component of the interpolator for α values from 1 to 5.

2.1.2 Sigmoid Interpolator at Transitions

A different approach can be used at a transition point, since the desired trajectory is a continuous curve. The sigmoid interpolator utilizes two sigmoids to create a composite function that sweeps the incoming and outgoing curves smoothly and

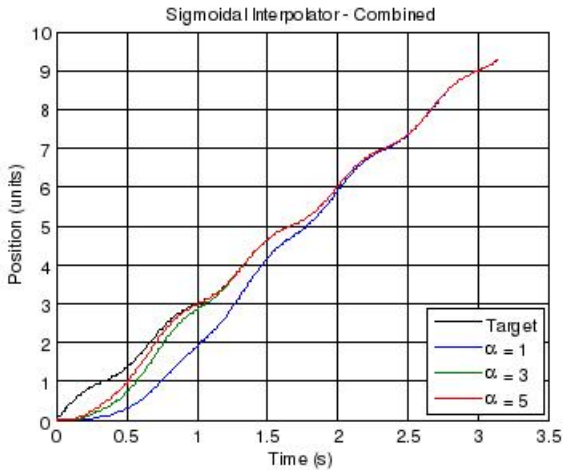

Figure 2.5: Combined Linear and Sinusoidal Exponential Interpolation.

asymptotically between 0 and 1. The sigmoid equation is:

$$S = \frac{1}{1 + e^{-\alpha(t-c)}} \tag{2.11}$$

where c is the transition point (in time) when the sigmoid value is equal to one, and α again is a time constant that affects the aggressiveness of the transitions. A positive α is used for the incoming curve and a negative one for the outgoing. At $t = c$, both curves contribute exactly half of their total value to the position at c, guaranteeing the correct transition point is reached. However, to assure a zero velocity at the transition point, which is imperative for direction switching, a correction term must be solved for and added to the sigmoid:

$$S_{in} = \frac{a}{a + e^{-\alpha(t-c)}} \tag{2.12}$$

$$S_{out} = \frac{b}{b + e^{-\alpha(t-c)}} \tag{2.13}$$

The coefficients a and b are the incoming and outgoing correction coefficients, respectively. The input and output trajectories each have a separate correction term that is

derived from two constraints to be enforced on the transition, firstly:

$$S_{in}(c) + S_{out}(c) = 1 \tag{2.14}$$

Next, the values of a and b must be selected to determine the blending of the two curves to be joined. If $a = b = 1$, there will be an even blending throughout the transition. For $a > b$, the sigmoid will transition after c, and $a < b$ will generate a transition prior to c. A blending ratio R can be selected at the midpoint c as:

$$R = \frac{S_{in}^{des}(c)}{S_{out}^{des}(c)} \tag{2.15}$$

Taking note of the fact that:

$$\begin{aligned} S_{in}(c) &= \frac{a}{1+a} \\ S_{out}(c) &= \frac{b}{1+b} \end{aligned} \tag{2.16}$$

we can achieve $S_{in}(c)/S_{out}(c) = R$ by choosing a and b as:

$$a = R \tag{2.17}$$

$$b = \frac{1}{R} \tag{2.18}$$

Note that these values are unaffected by α and that velocity and acceleration may increase even if the outgoing trajectory is decreasing. Also, it is unnecessary to perform this correction on trajectories continuing in the same direction. An example of a sigmoid calculated in this manner is shown in Figure 2.6. Using these S_{in} and S_{out} values, an acceleration and jerk-limited curve going smoothly through the transition point and not gouging the part in any portion of the transition, as shown in Figure 2.7.

As for the sinusoidal component of the path, care must be taken so that the oscillation does not gouge the part when nearing transition points. To prevent this, using an overlaying (2.12) and (2.13) with a c value away from the transition point coupled with an a or b value that is very small (but not zero) will remove the sine component from the transition trajectory, and may also be used to end a move.

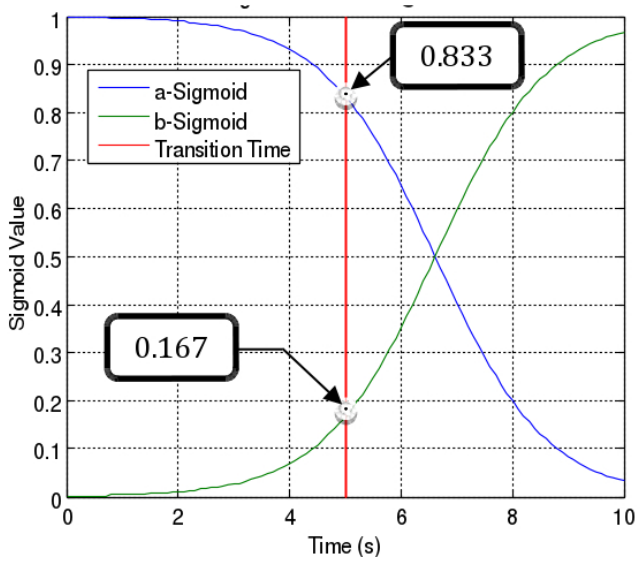

Figure 2.6: Corrected Sigmoid values at transition. The blue curve is the interpolated value of the incoming segment, and the green curve is the outgoing one.

2.2 Exponential and Sigmoidal Interpolation Results

As shown in Figures 2.9 and 2.8 below, at low acceleration limits the two methods of trajectory interpolation produce vastly different results, with the exponential/sigmoidal (E/S) interpolator converging to the desired trajectory quickly. Figures 2.11 and 2.10 show that tracking is similar between the two at high acceleration limits, but a time delay is still apparent in the linear interpolator.

2.3 Quantitative Analysis

To provide a quantitative comparison of performance between the two interpolation techniques, a performance index, J, has been defined. J is comprised of a weighted sum of penalties for tracking error and tool acceleration over the entire machining

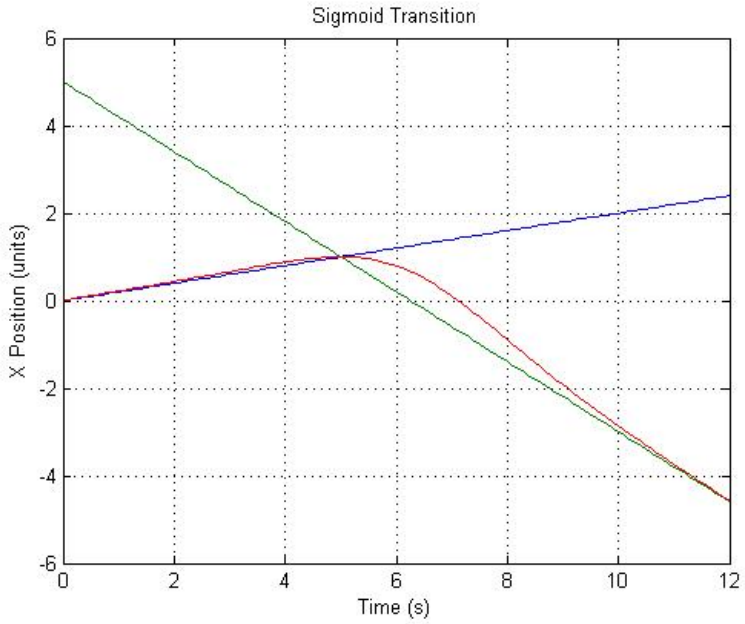

Figure 2.7: Sigmoid transition at $t = 5$, from a 1:5 slope to a -1:1 slope. Note that the sigmoid curve does not exceed the intersection of the two lines.

trajectory. This leads to an index of the form:

$$J = \int_0^T [k_e(y_{des} - y_{act})^2 + k_a \ddot{y}^2] dt \tag{2.19}$$

T is the run time, y_{des} is the target position, y_{act} is the interpolated value, and $\ddot{y}$ is the instantaneous acceleration value. Constants k_e and k_a are the weighting coefficients for the error tracking and acceleration. The scalar k_e was chosen to be $6 \times 10^2 \frac{in}{s-in^2}$, and $k_a \frac{s}{in}$ was set to 1. For comparison, the ramped sinusoid chosen was of slope 3, an amplitude of 0.15, a frequency of 2 Hz.

2.3.1 Error Tracking

By itself, the error tracking component of the performance index showed a notable improvement by using the E/S interpolator over the linear interpolator. As shown in Figure 2.12, the E/S interpolator showed a large advantage in tracking error at low acceleration limits, but as these limits are increased, that edge lessens significantly.

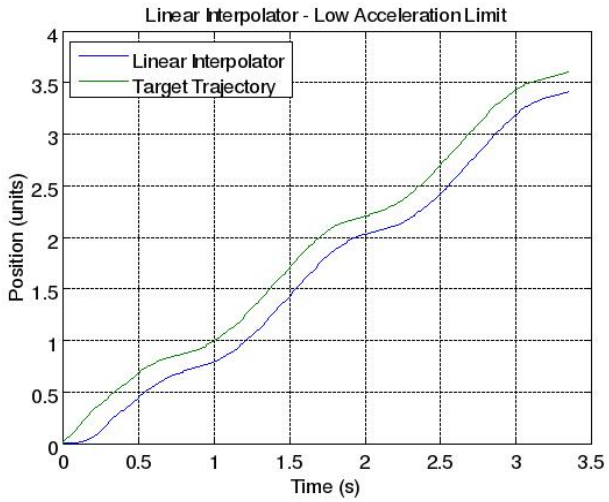

Figure 2.8: Low acceleration limit - Linear Interpolator.

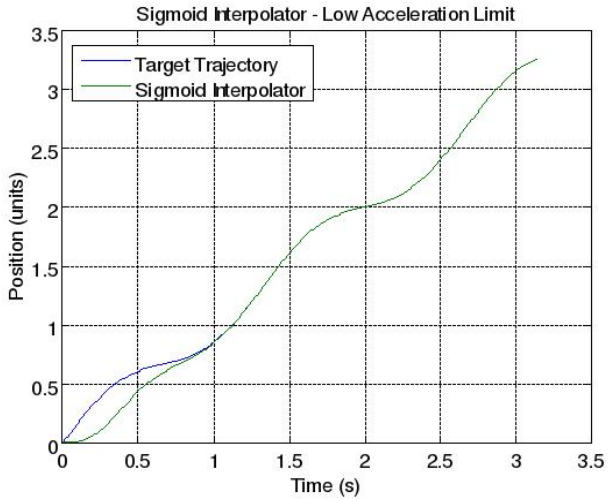

Figure 2.9: Low acceleration limit - Exponential Interpolator.

2.3.2 Acceleration Costs

As both of these interpolators are tracking similar curves, it can be expected that they perform similarly. However, as acceleration limits are increased, the E/S interpolator tends to dedicate less effort to acceleration. This can be attributed to the linear interpolator always choosing maximum positve/negative jerk when it is changing feed speeds. The result is shown in Figure 2.13.

These two indices are then weighted by their respective terms, k_e and k_a, and the overall improvement of the E/S interpolator can be seen in Figure 2.14.

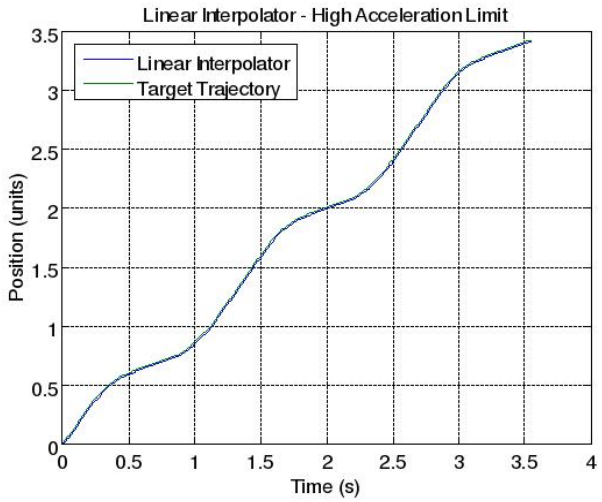

Figure 2.10: High acceleration limit - Linear Interpolator.

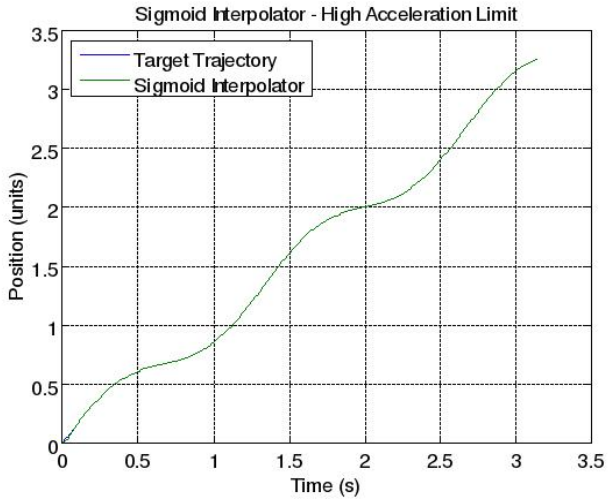

Figure 2.11: High acceleration limit - Exponential Interpolator

2.3.3 Alpha Value

The tracking aggressiveness for the E/S interpolator is determined by the α value. Plotting α vs. J generates the curve shown in Figure 2.15, and shows a minimum-cost value for α can be determined. At this α value, J is less than one third of the value for the linear interpolator.

2.3.4 Sigmoid Transition

The E/S interpolator also performed well at transition points. For a 10 inch movement back and forth at 10 in/s, with a 500 in/s^2 maximum acceleration, the linear

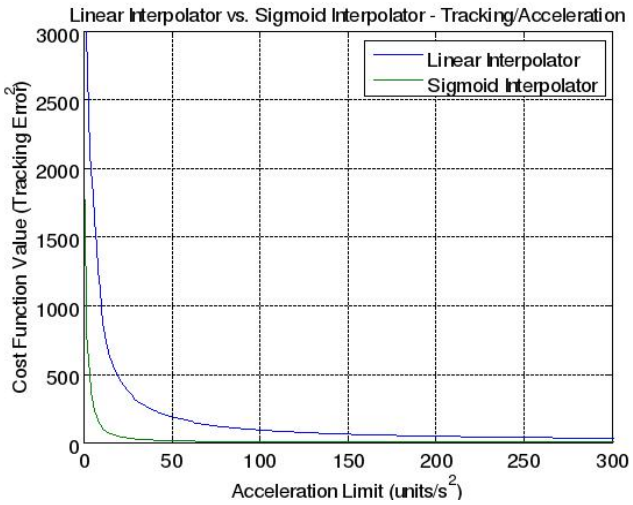

Figure 2.12: Tracking error comparison. The upper blue curve is the Linear Interpolator and the lower green curve is the E/S interpolator.

interpolator displayed its characteristic tracking error issues. For the E/S interpolator, the maximum allowable α was determined to be 50. Using the same function and coefficients for J, the E/S interpolator was 30 times more efficient at tracking the target trajectory and decreasing the acceleration used over the entire movement. See Figures 2.16 and 2.17 for detail.

2.4 Exponential/Sigmoidal Interpolator Conclusion

In this section, an alternative interpolation strategy was proposed to replace traditional jerk-limited techniques. The E/S interpolator was derived and shown to honor the physical constraints imposed by machining, i.e., jerk/acceleration/velocity/position, and to navigate segment transitions without overshooting or gouging the part. Based upon a performance index consisting of tracking error and acceleration penalties, the E/S interpolator showed up to a 60% improvement over linear interpolation for low acceleration limits, and up to a 12% improvement at high acceleration limits. Using a "there and back again" movement and transistion, the E/S interpolator showed a 30-fold improvement over the linear interpolator at the same maximum acceleration limit.

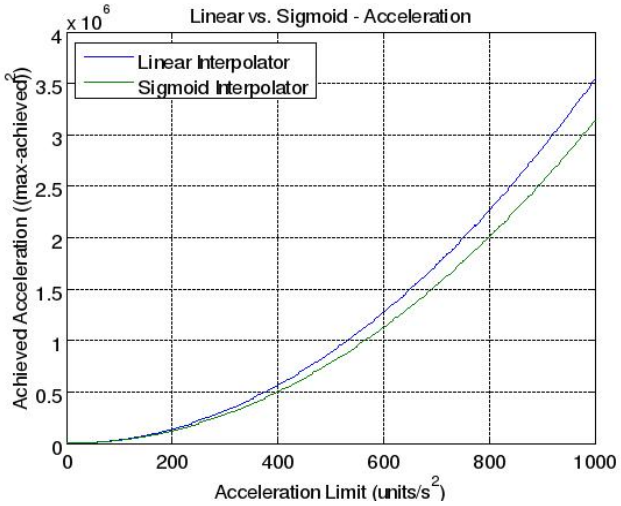

Figure 2.13: Achieved acceleration. The upper blue curve is the Linear Interpolator and the lower green curve is the E/S interpolator.

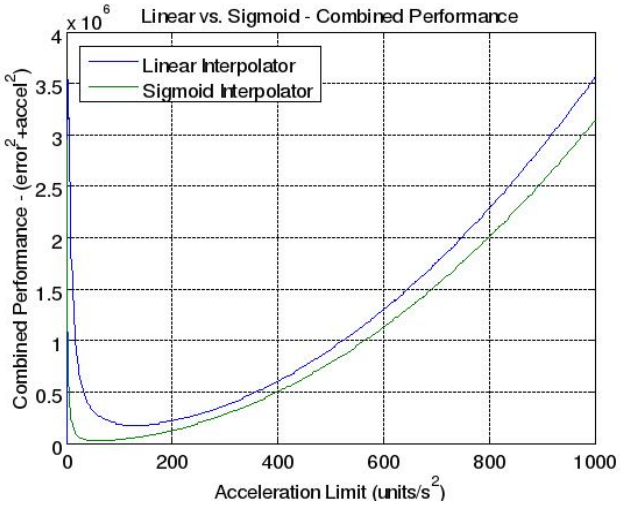

Figure 2.14: Combined tracking error and acceleration performance indices for both interpolators.

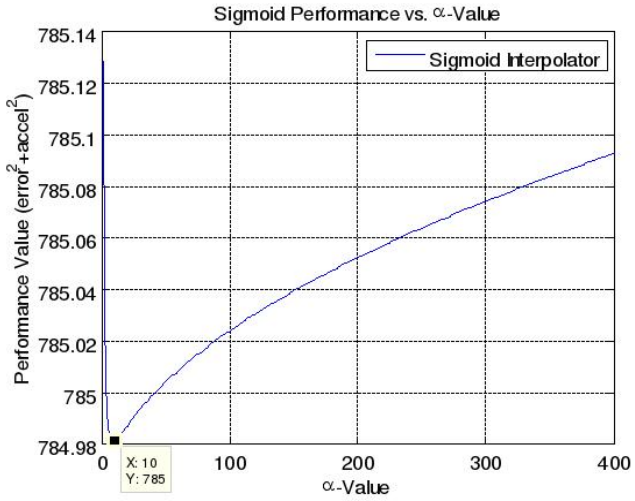

Figure 2.15: E/S interpolator α value versus J.

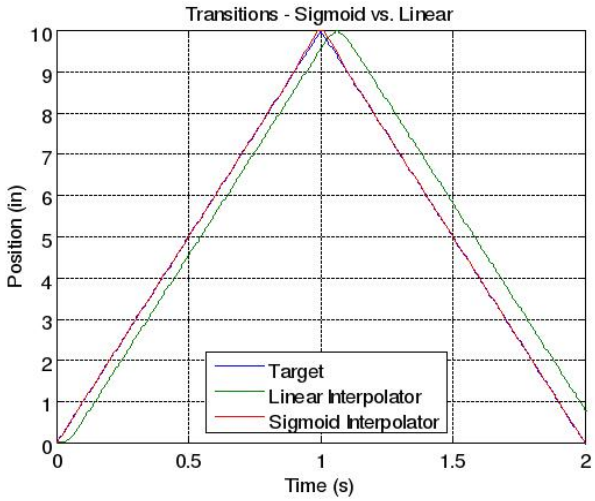

Figure 2.16: Interpolator comparison at transition.

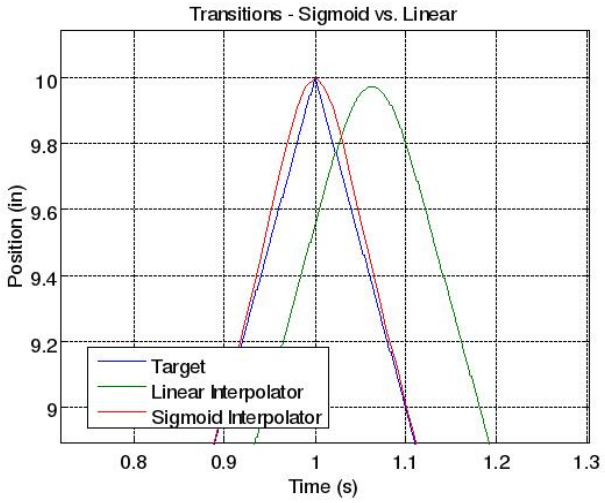

Figure 2.17: Interpolator comparison at transition, zoomed in.

CHAPTER 3: STANDARD MACHINE TOOL CONTROL STRATEGIES

In order for MTP to function as described, the machine tool controller must be able to accurately reproduce a trajectory of multiple ramped sinusoids. The two characteristic manners in which machine tools fail to achieve this are in amplitude falloff, which can also be seen in that figure, and frequency falloff [3], due to the reasons stated in the previous chapter. The tool path typically consists of multiple different ramped sinusoids with different ramp rates and oscillation frequencies that are knitted together. One example of this is during facing of a cylindrical part: as the radial distance varies, so must the feed rate and thus the MTP parameters. Precise tracking of a prescribed *trajectory* (i.e., tool position versus time) is extremely important in these types of cases.

Ultimately, two key requirements must be met for successful control of MTP processes:

1. Transient and steady-state tracking errors must be kept below a very tight machining tolerance.

2. The process must be repeatable over hundreds of iterations, *under variable environmental conditions.*

3.1 Time-Domain Control

The first method for tracking the desired reference trajectory is within the time-domain. While it is *possible* for a machine tool to operate in an open-loop configuration with very well known plant parameters, for the sake of quality control, most machine tools make use of closed-loop controllers as this provides an error signal for

the controller to track the performance of the tool. The predominant closed- loop control strategy used in machine tools is the Proportional-Integral-Derivative (PID) controller [4], shown in Figure 3.1.

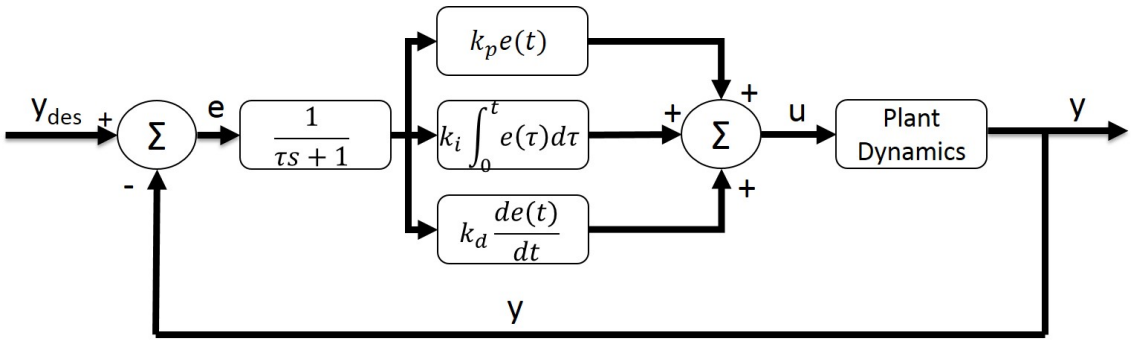

Figure 3.1: PID control system block diagram.

PID control has four separate components that can be used to aid in tracking the reference signal, y_{des}. The error signal, e, first makes its way through a filter with time constant τ. It then branches into the three PID contributers to the control signal, u: a proportional section ($k_p e(t)$), an integral section ($k_i \int_0^t e(\tau)d\tau$), and the derivative section ($k_d \frac{de(t)}{dt}$). Each has their own tunable coefficient (k_p,k_i,k_d) that must be determined. The control signal processes through the plant dynamics and generates the output y. While reasonable performance can be obtained, standard time-domain control techniques without learning will typically lead to consistently repeated transient errors that exceed the part tolerance, thereby failing to achieve the first aforementioned requirement. For example, if one has a PID controller driving a motor and encoder system (i.e., a feedback system), while the system may function within specifications, there still is a predictable amount of error in the system's response to a driving reference signal due to the system's time constants (exaggerated in Figure 3.2 for understanding) and deterministic nature. Other time-domain control strategies will exhibit similar issues. H_∞ methods, which attempt to minimize the H_∞ norm of the plant's closed-loop transfer function, cannot correct transient error or even

steady-state error. Model predictive control (MPC), which re-optimizes the trajectory over a set time horizon, suffers the same ails. Even Internal model principle (IMP)-based control techniques, which can embed sinusoidal trajectories in the controller such that it is tracking the deviation from a known sinusoid [6], has no method to correct transient errors. None of these control strategies has any method to correct for iteration-domain varying disturbances, which can be seen in Figure 3.3 over 500 iterations. To remedy these concerns, we must look into the iterative domain.

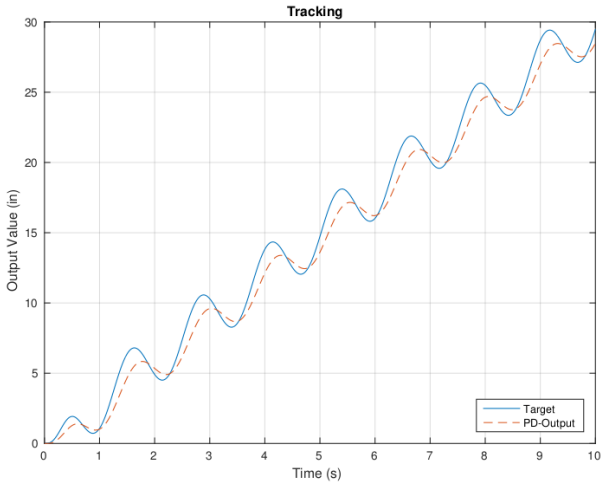

Figure 3.2: PID control of an exponentially growing ramped sinusoid MTP trajectory.

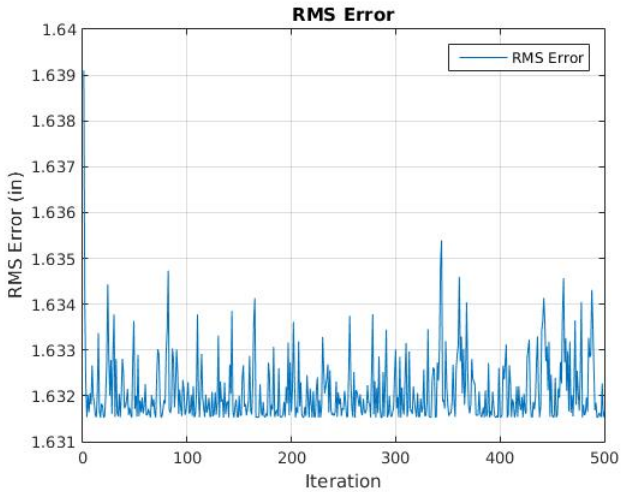

Figure 3.3: PID control with iteration-varying disturbances over 500 iterations.

3.2 Iterative Domain Control

In response to the concerns over repeated transient errors and the availability of multiple machining iterations to learn from, iterative learning control (ILC) stores the error and control input trajectories from the previous iteration and manipulates them to correct the control actuator input for the next iteration [12]. Use of ILC was pioneered by Arimoto [15] in 1984, and has been extended by Moore and Chen [16], Alleyne [13], Hoelzle and Barton [20], and many others. The implementation of ILC is as varied as standard controller design, and also has similarities to those control structures. For example, some familiar techniques for both feedback controller and ILC design include proportional-integral-derivative (PID), H_∞, and cost-bound (linear regression) [13].

ILC has proven to be a very effective control technique in eliminating transient and steady-state error over repeated iterations. However, even learning algorithms can increase error or even destabilize a system, as most variants of ILC do not have a method for handling disturbances that vary randomly from one iteration to the next, which can lead to improper control inputs being generated. For example, a previously learned control modification for a high value of disturbance will result in an incorrect control input when the following iteration's disturbance is low. This becomes problematic when random disturbances (arising, for example, through temperature variations) over the machining of hundreds of parts can lead to widely differing part tolerances and surface finishes.

This can be clearly seen in the use of Proportional-Derivative ILC (PD-ILC) as developed by Arimoto [15]. Using the SISO discrete-time system dynamics in Equation

(3.1):

$$\begin{aligned} \underline{x}(t+1) &= A(d_i)\underline{x}(t) + B(d_i)u(t) \\ y(t) &= C(d_i)\underline{x}(t) \\ e(t) &= y_{des}(t) - y(t) \end{aligned} \tag{3.1}$$

with A, B, C as the state space matrices, $\underline{x}$ as the system states, and d_i as the iteration-varying disturbance. The system diagram is shown in Figure 3.4, and the prototypical definition of discrete-time PD-ILC is given by:

$$u_{i+1}(t) = u_i(t) + k_p e_i(t+1) + k_d(e_i(t+1) - e_i(t)), \tag{3.2}$$

where i is the iteration index, k_p is the proportional gain coefficient, k_d is the derivative gain coefficient, and e is the error. The control input augmentation for each iteration is determined based on the error and error rate from the previous iteration.

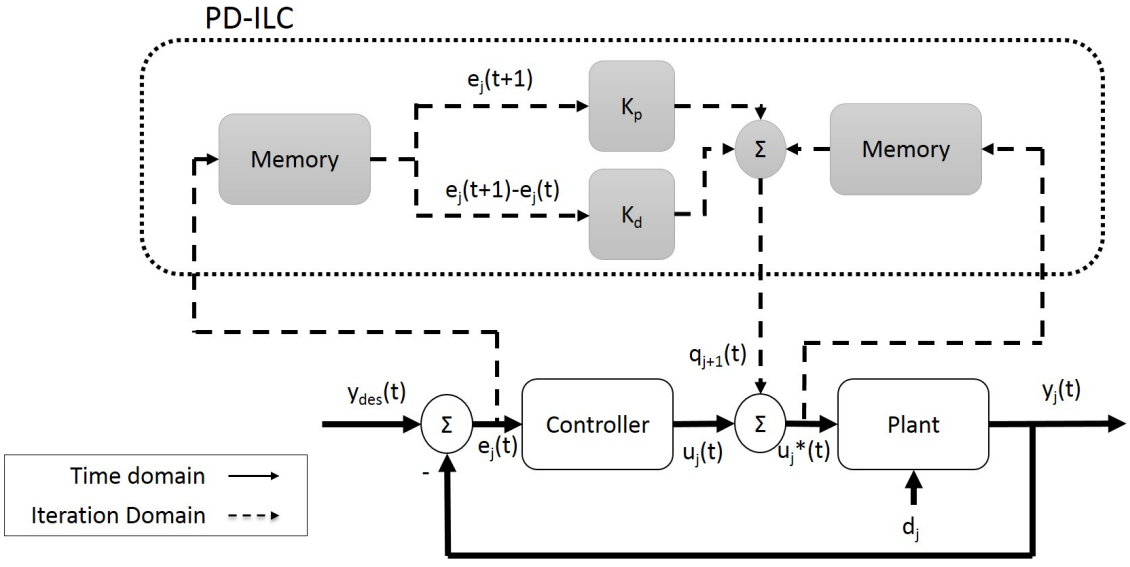

Figure 3.4: PD-ILC system block diagram.

To determine the values of k_p and k_d that will sufficiently guarantee monotonic convergence, we must analyze the plant Markov impulse parameters, given by:

$$p_m = CA^{m-1}B \tag{3.3}$$

where p is the Markov parameter, and m is the parameter index. With these values, Norrlof and Gunnarsson [17] determined that a sufficient condition for asymptotic stability and Lee and Bien [21] showed monotonic convergence is satisfied by:

$$|1 - (k_p + k_d)p_1| < 1 \tag{3.4}$$

As long as Equation 3.4 is satisfied, k_p and k_d may be selected to suit application-specific needs. For constant or zero disturbances, this can result in desired monotonic convergence as expected (see Figure 3.5. Unfortunately, one those disturbances begin to vary in the iteration domain, PD-ILC is incapable of improving its performance after a certain limit (see Figure 3.6).

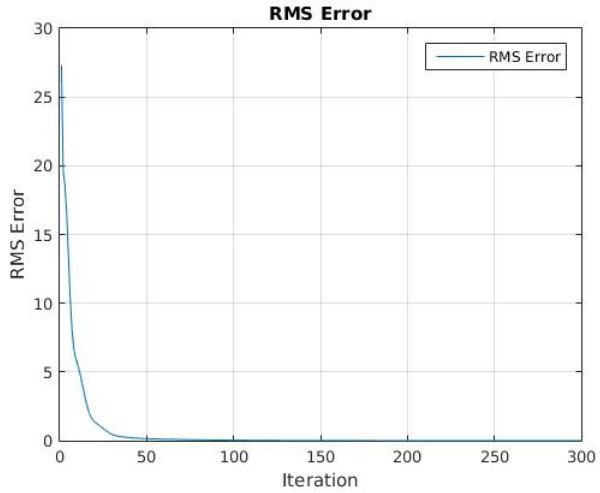

Figure 3.5: A typical application of PD-ILC over 500 iterations with zero/constant disturbance.

The main reason for these symptoms is the lack of any determination of the relevance of the current disturbance to the previous iteration's disturbance. This can easily be seen by a thought experiment: if, for a given system, the desired output was 5 inches of positive movement in the presence of disturbance level 5. An ILC system

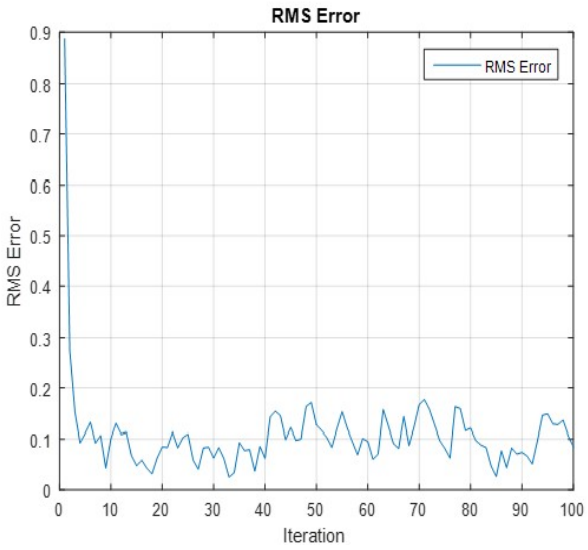

Figure 3.6: A typical application of PD-ILC over just 100 iterations with an iteration-varying disturbance.

would converge nicely while the disturbance maintained a value of 5. However, if it suddenly jumped to 10, or worse, -5, how relevant would the control input be? Obviously, some percentage would be correct, but there would be some error due to the jump in the disturbance. Once the disturbance varies on every iteration, a result such as Figure 3.6 appears.

Flexible ILC, a library-based version of ILC pioneered by Hoelzle and Barton [20], represents an initial effort toward remedying this issue. Flexible ILC stores disturbance values and maps the optimal control input through plant inversion and assumed knowledge of the mapping of disturbance to plant dynamic response. If a new disturbance is encountered, it then interpolates between the nearest tabulated disturbance values to determine the proper control input. While flexible ILC represents an important step toward the consideration of external disturbances through ILC as well as library-building, the framework of [20] can be further enhanced through three important measures:

1. The interpolation algorithm can be extended to consider more than just the nearest two tabulated disturbances; in this manner, the ILC strategy can make

use of more than just two past trajectories in its learning.

2. When determining how to weight past control input trajectories, both the disturbance *and* resulting performance can be taken into account. For example, if two previous iterations share the same disturbance value but different levels of performance, the higher-performing control input trajectory should be weighted more heavily in the ILC iteration.

3. Finally, little if any information should be required regarding the plant and the mapping of the disturbance to the plant response.

A strategy to apply both of these measures is supplied in the following chapter.

CHAPTER 4: DISTURBANCE AND PERFORMANCE-WEIGHTED ITERATIVE LEARNING CONTROL

Given the limitations of time-domain and current ILC techniques, this thesis proposes a new ILC technique, termed Disturbance and Performance-Weighted ILC (DPW-ILC), which accounts for disturbances that vary from one iteration to the next *and* the tracking performance attained under these disturbances. DPW-ILC requires little a priori knowledge of plant dynamics or mapping of a disturbance to the plant dynamics. It is assumed, as in all ILC implementations, that the time-domain system is stable and that reference trajectory is unchanging. This algorithm stores each iteration's control inputs, error signals, and measured disturbance values, then uses a weighted metric to determine the relevance of each previous iteration to the current iteration. This metric is based on error signal, error rate, and the similarity of the disturbance values. Previous iterations with good tracking performance and similar disturbance values to the currently measured disturbance are weighted more heavily. The DPW-ILC algorithm in this paper is built on Arimoto's PD-ILC [15], but can be generalized to other control methodologies. The DPW-ILC algorithm is validated on an MTP example where thermal expansion results in significant plant variations from one iteration to the next. The results presented herein demonstrate a marked improvement between basic PD-ILC and DPW-ILC under this scenario.

4.1 DPW-ILC Framework

The fundamental structure of DPW-ILC is based on two central tenets:

1. When learning from previous iterations, one should apply more weight to iterations where the disturbance was *similar* to the presently-measured disturbance.

2. When learning from previous iterations, one should apply more weight to iterations where *good* tracking performance was realized.

The overall structure of DPW-ILC, which takes into account these two tenets, is given in Figure 4.1.

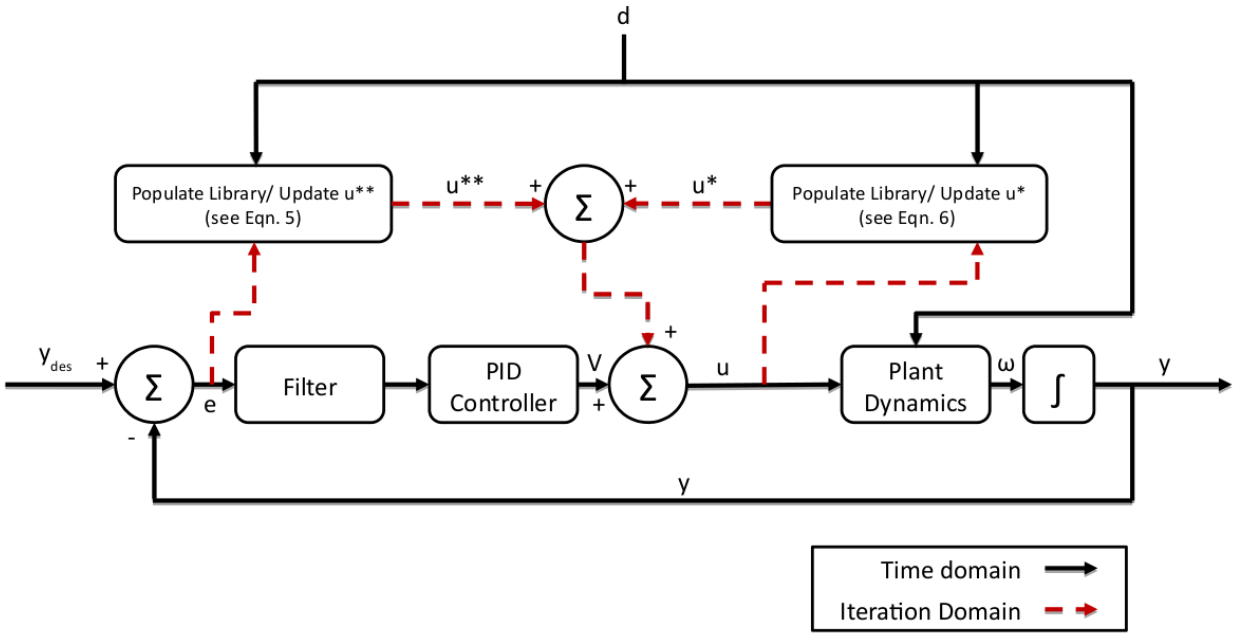

Figure 4.1: DPW-ILC standard layout

In describing the mathematical framework behind DPW-ILC, we will consider a SISO system with plant dynamics in discrete time given by:

$$\begin{aligned} \underline{x}(t+1) &= A(d_i)\underline{x}(t) + B(d_i)u(t) \\ y(t) &= C(d_i)\underline{x}(t) \\ e(t) &= y_{des}(t) - y(t) \end{aligned} \tag{4.1}$$

where A, B, and C are the discrete-time plant matrices, $\underline{x}$ are the system states, u is the system input, and y is the system output, y_{des} is the desired output, and t is the time index. The value d_i represents the environmental disturbance signal at the

present iteration, denoted by i. The disturbance is assumed to vary from one iteration to the next but is approximated as constant over the course of an iteration. For this system, the ILC law is given by:

$$u_{i+1}(t) = f(E_i, D_i, t) + g(U_i, D_i, t), \tag{4.2}$$

where the subscript $i+1$ represents iteration $i+1$ and E_i, U_i, and D_i represent historical error, control input, and disturbance trajectories that are stored within a library that is updated at every iteration. The stored error, control input, and disturbance trajectories comprise all past iterations and are therefore given by:

$$\begin{aligned} U_i &= \left[\underline{u}_{i-1}\ \underline{u}_{i-2}\ \dots\ \underline{u}_{i-m}\right], \\ E_i &= \left[\underline{e}_{i-1}\ \underline{e}_{i-2}\ \dots\ \underline{e}_{i-m}\right], \\ D_i &= \left[\underline{d}_{i-1}\ \underline{d}_{i-2}\ \dots\ \underline{d}_{i-m}\right], \end{aligned} \tag{4.3}$$

where m is the total number of iterations that have been completed and $\underline{u}_i$, $\underline{e}_i$, and $\underline{d}_i$ represent the control input, disturbance, and error *trajectories*, respectively, at iteration i:

$$\begin{aligned} \underline{u}_i &= \left[u_i(0)\ u_i(1)\ \dots\ u_i(N-1)\right]^T, \\ \underline{e}_i &= \left[e_i(0)\ e_i(1)\ \dots\ e_i(N-1)\right]^T, \\ \underline{d}_i &= \left[d_i(0)\ d_i(1)\ \dots\ d_i(N-1)\right]^T. \end{aligned} \tag{4.4}$$

In the above, N is the number of discrete time steps in each iteration.

In arriving at expressions for $f(E_i, D_i, t)$ and $g(U_i, D_i, t)$ in equation (4.2), the central idea is to apply more weight to past iterations with similar disturbances and good performance. This is accomplished through the following general weighted expressions

for $f(E_i, D_i, t)$ and $g(U_i, D_i, t)$:

$$f(E_i, D_i, t) = u^{**}(t) = \Gamma_1 \sum_{j=1}^{m} w_j(\underline{N_i}) f^{'}(\underline{e}_i, t), \tag{4.5}$$

$$g(U_i, D_i, t) = u^{*}(t) = \Gamma_2 \sum_{j=1}^{m} w_j(\underline{N_i}) g^{'}(\underline{u}_i, t) \tag{4.6}$$

where $w_j(\underline{N_i})$ is a weight assigned to each previous iteration's error and control signal based on its relevance, $\underline{N_i}$, to the current iteration. These weights are taken to be the same for both control and error signal, and they are assumed to sum to 1 across all iterations. The "learning functions," $f^{'}(\underline{e}_i, t)$ and $g^{'}(\underline{u}_i, t)$, are presently expressed as generic functions of the error and control signal trajectories, respectively. Finally, Γ_1 and Γ_2 are user-defined learning gains. At this point in the derivation, two main tasks are required to fully define the DPW-ILC formulation:

1. The form of $f^{'}(\underline{e}_i, t)$ and $g^{'}(\underline{u}_i, t)$ must be defined.

2. The form of the weighting functions, $w_i(\underline{N_i})$, must be defined.

4.1.1 Defining Learning Functions Using PD-ILC

In this work, we will use Arimoto's PD-ILC [15] in deriving expressions for $f^{'}(\underline{e}_i, t)$ and $g^{'}(\underline{u}_i, t)$. Using the system dynamics established above in Equation 4.1, the prototypical definition of discrete-time PD-ILC is given by:

$$u_{i+1}(t) = u_i(t) + k_p e_i(t+1) + k_d(e_i(t+1) - e_i(t)), \tag{4.7}$$

where i is the iteration index, k_p is the proportional gain coefficient, k_d is the derivative gain coefficient, and e is the error. The control input augmentation for each iteration is determined based on the error and error rate from the previous iteration. Within the PD-ILC framework of equation (4.7), $f^{'}(\underline{e}_i, t)$ and $g^{'}(\underline{u}_i, t)$ are given by:

$$
\begin{aligned}
f^{'}(\underline{e}_i, t) &= k_p e_i(t+1) + k_d(e_i(t+1) - e_i(t)), \\
g^{'}(\underline{u}_i, t) &= u_i(t).
\end{aligned}
\tag{4.8}
$$

To determine the values of k_p and k_d that will sufficiently guarantee monotonic convergence, we must analyze the plant Markov impulse parameters, given by:

$$p_m = CA^{m-1}B \tag{4.9}$$

where p is the Markov parameter and m is the parameter index. With these values, Norrlof and Gunnarsson [17] determined a sufficient condition for asymptotic stability and Lee and Bien [21] showed monotonic convergence is satisfied by:

$$|1 - (k_p + k_d)p_1| < 1 \tag{4.10}$$

As long as Equation 4.10 is satisfied, k_p and k_d may be selected to suit application-specific needs.

4.1.2 Defining Weighting Functions Based on Performance and Disturbance Similarity

The central concept of DPW-ILC lies in weighting previous iterations based on the similarity of their disturbance to the currently measured disturbance and the quality of tracking performance achieved. This is achieved by computing a *relevance* index that quantifies both of the aforementioned measures.

In quantifying disturbance similarity, the current iteration's measured disturbance d_i is compared to all the other stored disturbances by taking the absolute value of the difference:

$$\underline{\widetilde{d}} = \left[|d_i - d_{i-m}| \ |d_i - d_{i-2}| \ \ldots \ |d_i - d_{i-1}|\right]^T \tag{4.11}$$

Note that the disturbance is assumed to remain constant over the course of a single iteration.

Performance relevance is computed simply by considering the tracking error and rate of change in tracking error for a given iteration. Combined with disturbance similarity, the overall relevance index between iteration i and j is given by:

$$N_{i,j} = \frac{1}{\int_0^T \left[N_e e_j(t)^2 + N_{er} \dot{e_j}(t)^2 + N_d \widetilde{d_j}^2 \right] dt} \tag{4.12}$$

where $N_{i,j}$ is the relevance index comparing the current iteration i to iteration j. N_e, N_{er}, and N_d are the weights for error, error rate, and disturbance distance relevance, respectively. These weights are user-selected and are designed balance the importance of disturbance similarity and tracking error. Once these index values have been determined for each previous iteration, they must be normalized such that the sum of all relevance indices is 1. As such, we determine the weighting values, $w_i(N_i)$, for each iteration as follows:

$$w_j(\underline{N_i}) = \frac{N_{i,j}}{\sum_{k=1}^{i-1} N_{i,k}}, \tag{4.13}$$

where $\underline{N_i}$ represents the full vector of relevance indices from $j = 1$ to $j = i - 1$, given by:

$$\underline{N_i} = \begin{bmatrix} N_{i,1} & N_{i,2} & \dots & N_{i,i-1} \end{bmatrix}^T. \tag{4.14}$$

4.2 Examples

The two following examples will help to demonstrate the implementation of the DPW-ILC algorithm. Both will make use of a core system which consists of a DC motor that controls the speed and position of a machine tool. The desired trajectory consists of an MTP process generated by the E/S interpolator for a 2-segment move.

In both examples, we will consider the impact of significant iteration-to-iteration temperature variations on the performance of the MTP process. Local temperature variations represent some of the most ubiquitous and consequential environmental disturbances on manufacturing processes, as evidenced for example in [19], thereby representing an appropriate disturbance source for the validation of DPW-ILC. In both application examples, the control strategy conforms to the block diagram of Fig. 4.1.

4.2.1 Case I - Motor Control with Thermally Varying Resistance

In this first example of DPW-ILC implementation, a DC motor with temperature-dependent resistance is considered. The motor is modeled through the following second-order continuous-time model:

$$\begin{aligned} J\dot{\omega} + B\omega &= Ki \\ L\frac{di}{dt} + R(T)i &= V - K\omega \end{aligned} \tag{4.15}$$

where J, B, K, $R(T)$, and L are defined as in Table 4.1. The dynamic variable i is the instantaneous current of the circuit and ω is the angular velocity of the motor. The tool position is given by y, and a filtered PD controller with feedback is used to track the input in the time domain. The PD controller gains are shown below in Table 4.2.

For this example, we will model a thermal disturbance that affects the electrical resistance of the motor, changing the time constant of the system. The resistance varies according to:

$$R(T) = R_0[1 + \alpha_R(T - T_0)] \tag{4.16}$$

R_0 is the starting resistance at the starting temperature, T_0, which are $R(T)$ from Table 4.1 and 0°C, respectively. α_R is the thermal resistance coefficient for copper

wire, approximately $3.9 \times 10^{-3}\ K^{-1}$. A filtered random disturbance is used, allowing the temperature to vary up to 100°C in either direction.

Table 4.1: Table of dynamic system variables

Variable	Value	Units	Description
J	0.01	$kg - m^2$	Moment of Inertia - Rotor
B	1	Nms	Viscous Friction Constant
K	1	$\frac{N-m}{A}$	Torque/Back EMF Constants
$R(T)$	1	$ohms$	Electrical Resistance
L	0.005	H	Electrical Inductance

Table 4.2: Table of PD controller gains

Variable	Value	Units	Description
P	10	$\frac{V}{in.}$	Proportional Gain
D	1	$\frac{V-s}{in.}$	Derivative Gain
τ_{filt}	0.01	s	Filter Time Constant

4.2.1.1 DPW-ILC Setup

The general layout of the DPW-ILC system is shown in Figure 4.1. While the algorithm is the same as laid out in Section 2 above, the coefficients for the PD-ILC core (k_p and k_d) must be calculated first using the criteria for monotonic convergence presented in Equations (4.9) and (4.10). After determining the Markov parameter $p1$ to be 1.6×10^{-3}, the values of $k_p = k_d = 1.284$ were selected. The relevance index coefficients must then be selected for use with Equation (4.12). Based on the units of the measured disturbance relative to the error values, as well as a desire to prioritize error over error rate, the following values were selected: $N_e = 10$, $N_{er} = 2$, $N_d = 0.5$. Both Γ values were set to 1.

4.2.1.2 Results

The implementation of DPW-ILC resulted in tracking the target signal with a high level of accuracy, improving as the iterations increased, as shown in Figure 4. This is made clear by the error signal, shown in Figure 5. On the controller side of the DPW-ILC system, it is still apparent that the control input is being affected even 400

iterations in Figure 6. Figure 7 provides a graphical depiction of the relevance indices at the last iteration of the simulation, i.e., $\underline{N_{500}}$. Clearly, a small number of iterations possess high relevance indices when compared to all other iterations. In fact, the three most relevant iterations are those that have a disturbance very close to that measured on this iteration *and* have low errors and error rates.

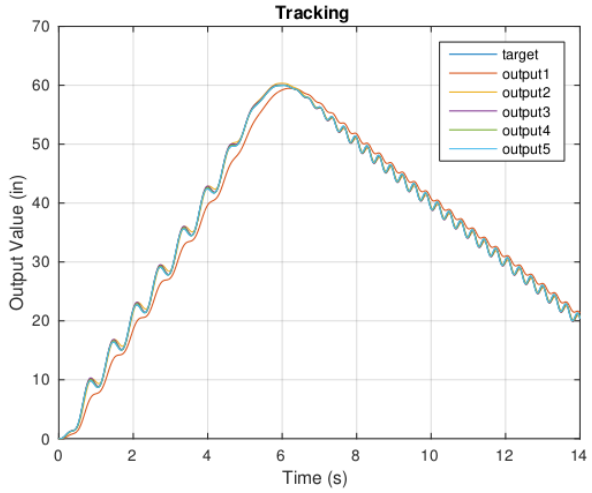

Figure 4.2: Thermal Resistance DPW-ILC example - Tracking. Only the initial PD iteration is high enough in error to be seen. All other iterations are very close to the target.

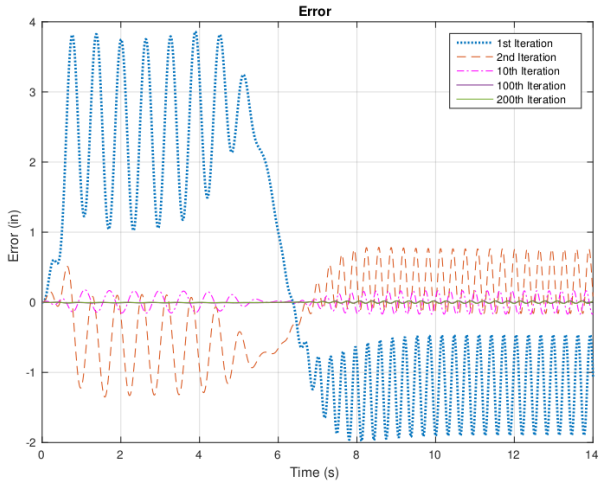

Figure 4.3: The values of error for several iterations for the Thermal Resistance Example.

The results (Figure 8) also show a dramatic improvement over non-ILC PID scheme

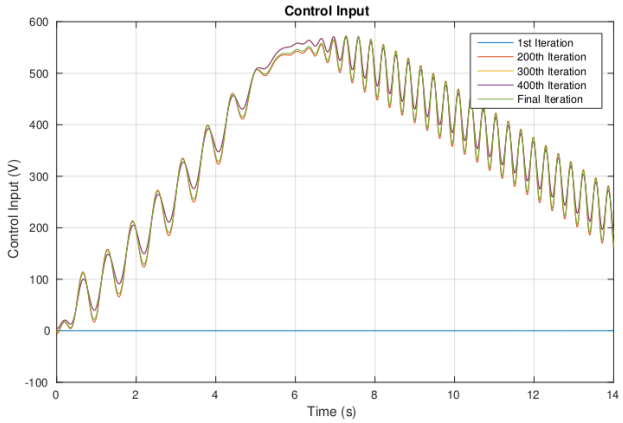

Figure 4.4: The Values of Control Input for several iterations for the Illustrative Example

and PD-ILC. PD-ILC does perform better than PID, but only to a point, whereas DPW-ILC shows continuing improvement over extended iterations, as well as minimizing the effect of randomized disturbances.

4.2.2 Case II - Nonlinear Thermal Expansion Example

In the second example, in the place of a thermal disturbance that influences resistance, the thermal variations act on a machine part that expands as the temperature increases. Furthermore, this thermal effect is modeled in a nonlinear fashion. As a result of this nonlinear thermal expansion, the actual position of the tool *relative to the part* (which is subject to thermal position), denoted by y, differs from the *absolute* tool position, denoted here by y^*. In this example, y^* is used for time-domain feedback, but it is assumed that an estimate of y is available after each iteration is complete, and therefore the error used to ILC learning between iterations is given by $y_{des} - y$. The temperature-dependent relationship between the part-relative position (y) and absolute tool position y^*, is given by:

$$y = \frac{y^*}{\alpha_1(T - T_0) + \alpha_2(T - T_0)^2 + 1} \tag{4.17}$$

α_1 and α_2 are two arbitrary thermal expansion coefficients, modeled as $3 \times 10^{-6}\ K^{-1}$ and $5 \times 10^{-7}\ K^{-2}$, respectively.

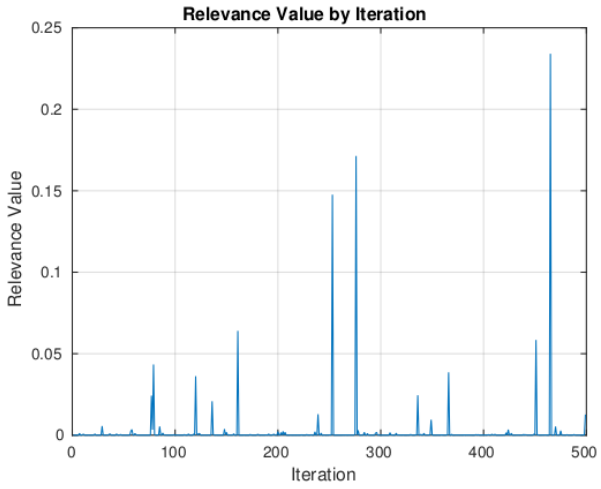

Figure 4.5: The final Relevance Index for the last iteration of 500 for Thermal Resistance example (i.e., the graph shows all of the elements of N_{500}). The handful of indices with high relevance values share a disturbance (temperature) that is similar to the measured temperature at iteration 500 *and* exhibit good tracking performance.

4.2.2.1 DPW-ILC Setup

All PD controller coefficients and motor variables are the same as in Tables 4.1 and 4.2. It follows that the Markov parameters and PD-ILC gains are similar. The N_e and N_{er} are the same, but in order to accommodate the large difference between the disturbance value of temperature and the small physical dimension of the disturbance effect of distance, N_d has been decreased to 10^{-4}. Both Γ values remain at 1.

4.2.2.2 Results

DPW-ILC tracks the target trajectory in a manner similar to the first example. The results in this example are less dramatic, as the disturbance-driven error is small, but still telling. While both variants of ILC show marked improvement over the PD only control, PD-ILC is unable to converge to zero error beyond a certain point when there is a changing disturbance applied to the system.

4.3 Conclusion

In this section, a library-based variant of PD-ILC called Disturbance & Performance-Weighted ILC that is capable of not only improving performance of a repeated task

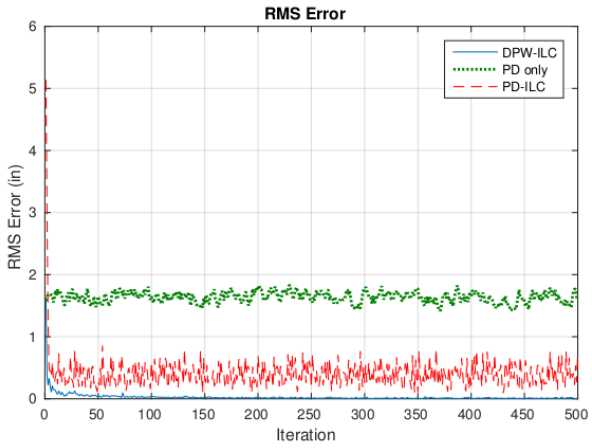

Figure 4.6: Comparison of RMS error values for DPW-ILC, PD-ILC, and PID Controls for Thermal Resistance Example.

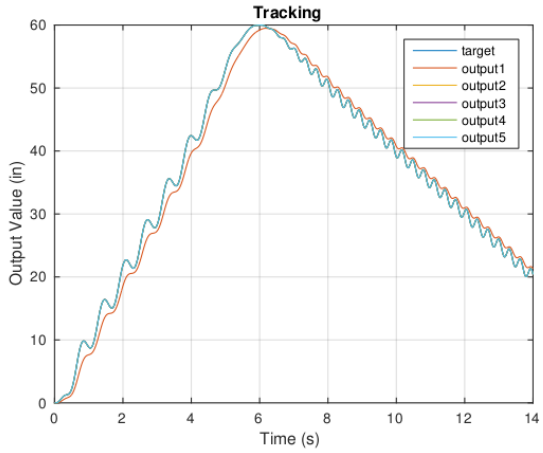

Figure 4.7: Tracking of DPW-ILC algorithm for Thermal Expansion example.

over time, but allowing the controller to respond better to iteration-varying disturbances such as a temperature fluctuation, was proposed. A general framework for DPW-ILC was laid out, allowing for the formulation of a system using PD-ILC as the core of ILC, while DPW-ILC managed the library and relevance weighting. DPW-ILC was demonstrated with an example for the variance of performance due to thermal resistivity and another due to non-linear thermal part expansion. In the thermal resistivity example, DPW-ILC showed a substantial improvement over basic PD control, as well as a much higher tolerance for disturbance variation over time compared

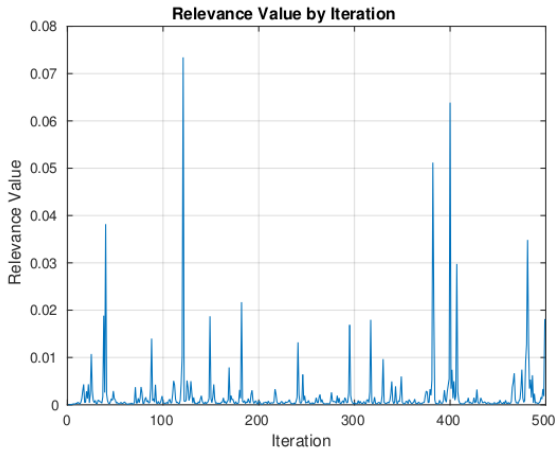

Figure 4.8: The final Relevance Index for the last iteration (500) for the thermal expansion example (i.e., the graph shows all of the elements of N_{500}). Similar to the first example, this last run has some high relevance values, which are iterations that have both close proximity to the current measured disturbance and a low error and error rate.

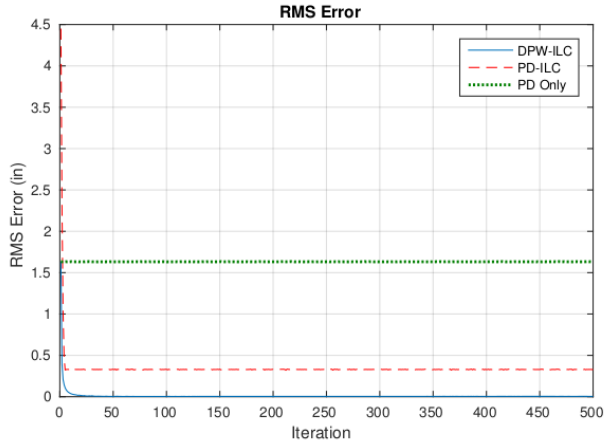

Figure 4.9: Comparison of RMS error values for DPW-ILC, PD-ILC, and PID Controls for Thermal Expansion Example

to PD-ILC. These trends continued through the nonlinear thermal part expansion demonstration, where DPW-ILC showed convergence to zero error, while both PD-only and PD-ILC hover at a disturbance-limited levels of error for continued iterations.

CHAPTER 5: CONCLUSION

In this thesis, weak links in the process of modulated tool path (MTP) machining were discussed, and two improvements to machine control strategies were developed. After an MTP reference trajectory is designed, it is imperative for the machine controller to precisely follow that trajectory.

The first step after the NC part program is determined is to interpolate the necessary positions of the tool at each time step, enforcing physical machine tool limitations. The jerk-limited linear interpolator, the mainstay method of this process for current machine tools, was shown to be unsuitable for oscillatory, or more specifically, ramped sinusoidal trajectories, due to segment delays and large acceleration demands. An alternative, called the Exponential/Sigmoid interpolator, was developed to aggressively minimize the tracking error and acceleration required by the control software. For the same MTP trajectory, the E/S interpolator was shown to have a 60% improvement over the linear interpolator at low accelerations and a 12% improvement at high accelerations. At transitions, the E/S interpolator demonstrated a 30-fold improvement over the jerk-limited linear interpolator for the same maximum acceleration limit.

After the target positions at each time step are determined, the machine tool must enforce these movements during machining time. The usual methods for achieving this were discussed, as well as their limitations. To improve upon the current systems, a library-based iterative learning control scheme called Disturbance and Performance-Weighted ILC was developed using a PD-ILC core. DPW-ILC was demonstrated using two examples: motor performance variance due to a thermal resistivity disturbance, and non-linear thermal expansion due to a temperature disturbance. In both exam-

ples, DPW-ILC showed marked improvements over both time-domain (PD) control and PD-ILC.

The combination of the E/S interpolator and DPW-ILC has shown tremendous potential for implementation in any machining scenario requiring high-accuracy movement for repetitive oscillatory tasks.

5.1 Future Work

There are several roads which can be traversed in order to improve the accuracy of MTP machining. First, the Exponential/Sigmoidal interpolator can likely be exploited to take advantage of the a and b values for different types of trajectories. Second, the E/S interpolator should be expanded out to multi-axis moves, which may prove somewhat challenging compared to linear techniques which scale easily. This interpolation technique will also need to be validated on a machine tool by someone with greater access to the internal workings of a machine tool controller.

With respect to the DPW-ILC scheme, a "trimming" mechanism could be implemented to remove previous iterations that have been replicated or contain relatively useless data. Any work along these lines will drastically improve the computation time between iterations, which can be lengthy as iterations exceed N_{500}. Also, the relevance index parameters ($N_e/N_{er}/N_d$) could be improved or replaced entirely with other parameters for better convergence. Lastly, DPW-ILC could be re-formulated to work with other versions of iterative-domain learning strategies to create novel alternatives.

BIBLIOGRAPHY

[1] Woody; Bethany A., Smith; Kevin Scott, Adams; David J., Barkman; William E., Babelay, Jr.; Edwin F., Methods and systems for chip breaking in turning applications using CNC toolpaths, U.S. Patent 8 240 234, issued August 14, 2012.

[2] L. Berglind, J. Ziegert, Chip Breaking Parameter Selection for Constant Surface Speed Machining, ASME 2013 International Mechanical Engineering Congress and Exposition, vol. 2B, pp. V02BT02A039, Nov 2013.

[3] Berglind L, Ziegert J, Lingerfelt K, Love W. Characterization of machine axis errors for modulated tool path (MTP) machining. In Transactions of the North American Manufacturing Research Institution of SME. January ed. Vol. 42. Society of Manufacturing Engineers. 2014. p. 630-636.

[4] Y. Altintas, Manufacturing Automation: Metal Cutting Mechanics, Machine Tool Vibrations, and Cnc Design, 1st ed., United Kingdom: Cambridge University Press, 2012, pp. 105-124.

[5] Rymansaib, Z., Iravani, P. & Sahinkaya, M. (2013). Exponential Trajectory Generation for Point to Point Motions. Wollongong, Austrailia, s.n., pp. 906-911.

[6] Costa-castello, R, et al. An Educational Approach to the Internal Model Principle for Periodic Signals. International Journal of Innovative Computing, Information and Control. V8, N8, pp. 5591-5606,August 2012.

[7] Y. Koren, C.C. Lo, Advanced Controllers for Feed Drives, CIRP Annals - Manufacturing Technology v41 n2, United Kingdom: Elsevier Ltd., 1992, pp. 689-698.

[8] J. Lee, Design of Controllers for Improving Contour Accuracy in a High-Speed Milling Machine, Ph.D. dissertation, Dept. Mech. Eng., Univ. of Florida, Gainsville, FL, 2005.

[9] G. Pelaez, Gu. Pelaez, J.M. Perez, A. Vizan, E. Bautista, Input Shaping Reference Commands for Cartesian Machines, Control Engineering Practice, V13, United Kingdom: Elsevier Ltd., 2004, pp. 941-958.

[10] R. Ramesh, M.A. Mannan, A.N. Poo, Tracking and Contour Error Control in CNC Servo Systems,International Journal of Machine Tools and Manufacture, V45, United Kingdom: Elsevier Ltd., 2005, pp. 301-326.

[11] M. Kim, S. Chung, A Systematic Approach to Design High Performance Feed Drive Systems, International Journal of Machine Tools and Manufacture, V45, United Kingdom: Elsevier Ltd., 2005, pp. 1421-1435.

[12] K.L. Moore, Iterative Learning Control for Deterministic Systems. London: Springer-Verlag, 1993.

[13] Bristow, D.A., Tharayil, M. and Alleyne, A.G., "A Survey of Iterative Learning Control," IEEE Control Systems, vol. 26, no. 3, pp. 96-114, 2006.

[14] M. Z. Md Zain, M. O. Tokhi, Z. Mohamed, M. Mailah, "Hybrid Iterative Learning Control of a Flexible Manipulator," presented at the 23rd IASTED Int. Conf. for Modeling, Identification, and Control, Grindelwald, Switzerland, Feb. 23-25, 2004, pp. 313-326.

[15] S. Arimoto, S. Kawamura, and F. Miyazaki, Bettering operation of robots by learning. J. Robot. Syst., V1, pp. 123-140, 1984.

[16] K.L. Moore, Y. Chen, V. Bahl, "Monotonically Convergent Iterative Learning Control for Linear Discrete-time Systems," Automatica, V41, N9, pp. 1529-1537.

[17] M. Norrlof, S. Gunnarsson, "Time and Frequency Domain Convergence Properties in Iterative Learning Control," Int. J. Contro., V75, N14, pp. 1114-1126, 2002.

[18] K.L. Moore, "Multi-Loop Control Approach to Designing Iterative Learning Controllers," presented at the IEEE Conference on Decision & Control, Tampa, FL, Dec. 1998, pp. 666-670.

[19] C.D. Mize and J.C. Ziegert, "Neural network thermal error compensation of a machinining center," Journal of the International Societies for Precision Engineering and Nanotechnology, V24, pp. 338-346, 2000.

[20] D.J. Hoelzle, K. Barton, "Flexible Iterative Learning Control Using a Library Based Interpolation Scheme," presented at the IEEE Conference on Decision & Control, Maui, HI, Dec. 2012, pp. 3978-3984.

[21] H.-S. Lee and Z. Bien, "Robustness and convergence of a PD-type iterative learning controller," Iterative Learning Control: Analysis, Design, Integration and Applications, Z. Bien and J.-X. Xu, Eds., Boston: Kluwer, 1998.

APPENDIX A: EXPONENTIAL/SIGMOID INTERPOLATOR GENERATION MATLAB CODE

```
1
2  %% Setup
3
4  % time step
5  delta = .01;
6
7  % feeds & speeds
8  x1 = 60;
9  x2 = -40;
10 v1 = 10;
11 v2 = 5;
12
13 % sine freq/amp
14 f1 = 10;
15 a1 = 2;
16 f2 = 20;
17 a2 = 1;
18
19
20 % Alpha Values
21 startAlpha = 5;
22 sigAlpha = 2;
23 sineAlpha = 3;
24 transitionPoint = 1; % 1 second offset
25
26 %% Generate line segments
27
28 %time vectors
29 tVec1 = 0 : delta : x1/v1;
30 tVec2 = tVec1(end)+delta:delta:tVec1(end)+abs(x2)/v2;
31 tVecTotal = [tVec1 tVec2];
32
33 xVec1 = zeros(1, length(tVecTotal));
34
35 for ii = 2:length(tVecTotal)
36
37     xVec1(ii) = xVec1(ii-1) + v1*delta;
38
39 end
40
41
```

```
42
43 xVec2 = zeros(1, length(tVecTotal));
44 xVec2(1) = xVec1(length(tVec1))+x1/v1*v2;
45
46 for ii = 2:length(tVecTotal)
47
48     xVec2(ii) = xVec2(ii-1) + sign(x2)*v2*delta;
49
50 end
51
52 %% Superimpose Sinusoids
53
54 wave1 = a1 * sin(f1*tVecTotal);
55 waveSig1 = sigmf(tVecTotal, [-sineAlpha tVec1(end)-1]);
56 xVec1 = xVec1 + waveSig1.*wave1;
57
58 wave2 = a2 * sin(f2*(tVecTotal-delta-tVec1(end))); % delta
     /tVec1 correction to start at 0
59 waveSig2 = sigmf(tVecTotal, [sineAlpha tVec1(end)+1]);
60 xVec2 = xVec2 + waveSig2.*wave2;
61
62
63 %% Sigmoid Transition
64
65 % extend segments
66
67 % xVec1Ext = xVec1(end)*ones(1,length(tVec2));
68 % xVec2Ext = xVec2(1)*ones(1,length(tVec1));
69 % xVec1 = [xVec1 xVec1Ext];
70 % xVec2 = [xVec2Ext xVec2];
71
72 % correction terms for sigmoid here
73 R = abs(v2/v1);
74
75 a = R;
76 b = 1/R;
77
78 sig1 =  a./(a + exp(sigAlpha.*(tVecTotal-tVec1(end))));
79 sig2 =  b./(b + exp(-sigAlpha.*(tVecTotal-tVec1(end))));
80
81
82 % generate sigmoid curves
83 %sig1 = sigmf(tVecTotal, [-sigAlpha tVec1(end)]);
84 %sig2 = 1 - sig1;
85
```

```
86 % correct with sigmoid
87 xVecSig = xVec1 .* sig1 + xVec2 .* sig2;
88
89 %% Startup Exponential
90 xVecComplete = xVecSig .* (1-exp(-startAlpha*tVecTotal.^2)
      );
91
92 % plot(tVecTotal, xVecComplete)
93 % grid on
94
95 x = xVecComplete;
96 t = tVecTotal;
```

APPENDIX B: SIMULINK MODELS FOR DPW-ILC EXAMPLES

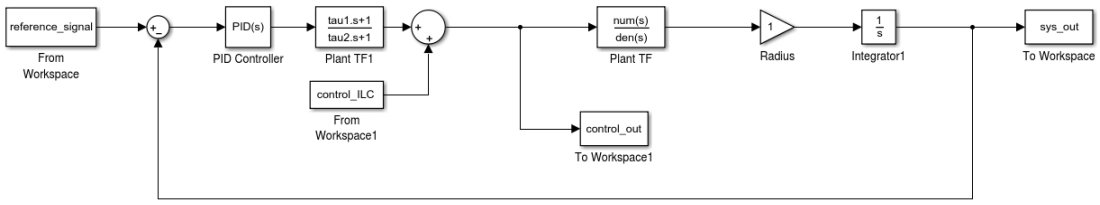

Figure B.1: Dynamic Model in MATLAB/Simulink for DPW-ILC Thermal Resistivity Example.

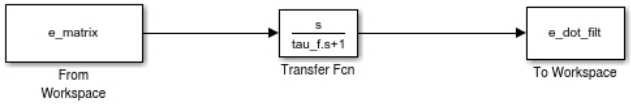

Figure B.2: Dynamic Model in MATLAB/Simulink for DPW-ILC Error derivative.

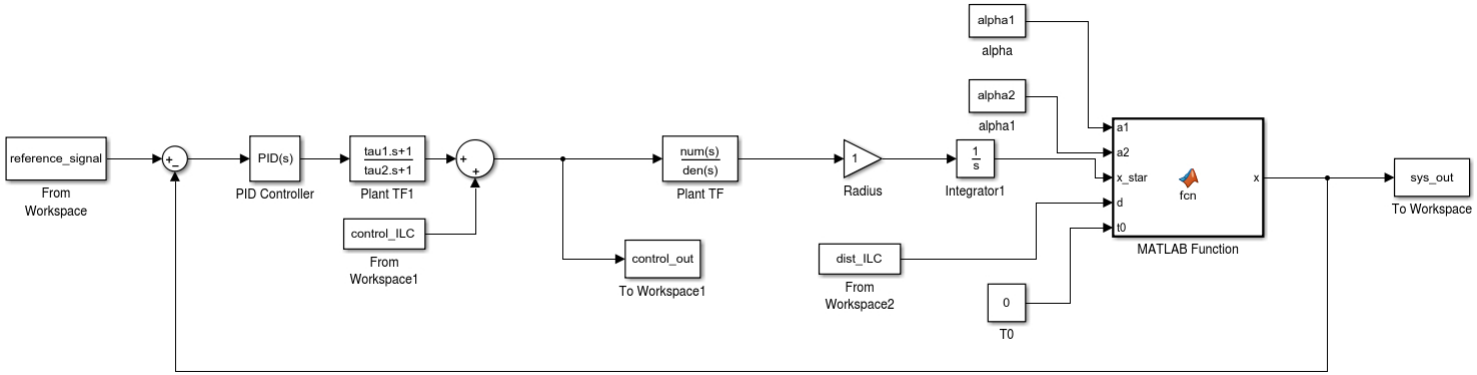

Figure B.3: Dynamic Model in MATLAB/Simulink for DPW-ILC Thermal Expansion Example.

APPENDIX C: DPW-ILC MATLAB CODE (FOR BOTH EXAMPLES)

```
% ********* NOTES *********
%   DPW-ILC - Variant
%   Variable types:
%   ALL_CAPS: program constants
%   camelHumps: internal variables
%   lower_delimited: SIGNALS to simulink
%
% ********** Clear Variables/Close Figures/Clear Command Window **********
clear all
close all
clc

% ********** Signal Generation - MUST RUN FIRST! **********

% Reference Signal Generation
sigGen % run signal generation program
reference_signal = [t' x'];
TIME_END = t(end);


% ********** Define Constants *********

% Simulation Constants (MATCH TO SIMULINK MODEL)
ITER = 10;
TIME_STEP = 0.01;
ZERO_TOL = 0.1;
Controller_P = 10;
Controller_I = 0;
Controller_D = 1;
Controller_N = 100;
tau1 = 0.1;
tau2 = 0.1;

% disturbance weighting
W = 1;
EPSILON = 1;
GAMMA_STAR = 1;
GAMMA_DOUBLE_STAR = 1;
COST_FNC_ERROR = 10;
COST_FNC_ERR_RATE = 2;
```

```
41 COST_FNC_DIST = 0.5;
42
43 % Plant Dynamics (30Nm/30A DC Motor)
44 J = 0.010; % Coulomb friction
45 B_0 = 1; % Viscous Damping Coefficient
46 K = 1; % Torque Constant
47 R_0 = 1; % Resistance
48 L = 0.005; % Inductance
49 T_0 = 0; % Start Temp
50
51 %**************************************
52 % Thermal Expansion Parameters
53 alpha1 = 3e-6;
54 alpha2 = 5e-7;
55
56 % Resistor Thermal Expansion Parameters
57 alpha_R = 3.9e-3;
58 R = R_0;
59
60
61 %% ********** System Initializations **********
62
63 % Generate Plant Dynamics and Variables for Simulink Model
64 s = tf('s');
65 sys = K/((J*s+B_0)*(L*s+R_0)+K^2); % system model
66 [plantNumerator, plantDenomenator] = tfdata(sys,'v'); %
     create plant transfer function
67 [A,B,C,D] = tf2ss(plantNumerator,plantDenomenator); %
     create SS vars
68 pDenStore=plantDenomenator(1); % save denomenator value
     for time constant shifting
69
70 sysInt = sys * (1/s);
71 sysIntDT = c2d(sysInt,0.01);
72 [DTNumerator, DTDenomenator] = tfdata(sysIntDT,'v');
73 [ADT,BDT,CDT,DDT] = tf2ss(DTNumerator,DTDenomenator);
74
75 % Kp and Kd for PD-ILC Portion
76 p1 = CDT*ADT^0*BDT; % 1st Markov Parameter
77 minK = 0/p1;
78 maxK = 2/p1;
79 kRange = maxK-minK-.1; % Range of K values
80 Kp = .001*kRange;
81 Kd = 10;
82
```

```
83  p1 = CDT*ADT^0*BDT; % 1st Markov Parameter
84  minK = 0/p1;
85  maxK = 2/p1;
86  kRange = maxK-minK-.1; % Range of K values
87  %Kp = .001*kRange;
88  %Kd = 10;
89
90  % control_ILC signal initialization
91  control_ILC = [t' zeros(1,length(reference_signal))'];
92  controlSignal = control_ILC(:,2)'; % control signal
        storage
93
94  % taskbar
95  wait = waitbar(0,'Initializing waitbar...');
96
97
98
99
100 %% ********** Iteration Program **********
101 for iteration = 1:ITER
102
103     percentage = iteration/ITER;
104     waitbar(percentage,wait,sprintf('Working... %d%%
            complete.',round(percentage*100)))
105
106     % ********** Disturbance Value/Signal Generation
            **********
107     distSignal = disturbanceVector(iteration) * ones(1,
            length(reference_signal));
108     disturbanceStore(iteration) = distSignal(end); %
            stores disturbance value
109     disturbanceNorm = abs(disturbanceStore -
            disturbanceStore(end));
110     dist_ILC = [t' distSignal'];
111     R = R_0 * (1 + alpha_R*(disturbanceStore(iteration)-
            T_0));
112     RStore(iteration) = R;
113
114
115     % ********** Performance Index Calculation **********
116     if iteration > 1
117
118         for itCount = 1:iteration-1
119
120             for tCount = 1:length(reference_signal)
```

```
121
122                 performanceErrorStore(itCount,tCount) =
                       COST_FNC_ERROR * (error(itCount,tCount)
                       .^2);
123                 performanceErrorRateStore(itCount,tCount)
                       = COST_FNC_ERR_RATE * (errorRate(itCount
                       ,tCount).^2);
124                 performanceDistStore(itCount,tCount) =
                       COST_FNC_DIST * ((disturbanceNorm(
                       itCount))).^2;
125
126             end
127
128             performanceStore(itCount,:) =
                   performanceErrorStore(itCount,:) +
                   performanceErrorRateStore(itCount,:) +
                   performanceDistStore(itCount,:);
129             performanceIter(itCount) = sum(
                   performanceStore(itCount,:));
130
131         end
132
133         performanceFlip = 1./(performanceIter+EPSILON);
134         performanceTilde = performanceFlip/sum(
               performanceFlip);
135
136     end
137
138     % DW-ILC Initialization for 1st iteration
139     if iteration == 1
140
141         u_star = zeros(1, length(reference_signal));
142         u_double_star = zeros(1, length(reference_signal))
               ;
143         u_star_store = u_star;
144         u_double_star_store = u_double_star;
145
146     else
147
148         % calculate for each iteration, iter-1 since have
               1 fewer value in distTilde, controlSignal-1
               since errorRate is difference valued
149         for itCount = 1:iteration-1
150
151             % calculate for each time step
```

```
152             for tCount = 1:(length(reference_signal)-1) %
                    -1
153
154                 u_star_store(itCount+1,tCount) =
                        GAMMA_STAR * (performanceTilde(itCount)
                        * (Kp.*error(itCount,tCount) + Kd.*(
                        errorRate(itCount,tCount))));
155
156
157             end
158
159             u_star_store(itCount+1,length(reference_signal
                    )) = 2*u_star_store(itCount+1,end-1) -
                    u_star_store(itCount+1,end-2);% arbitrary
                    EOL correction, slope based
160
161             for tCount = 1:length(reference_signal)
162
163                 u_double_star_store(itCount+1,tCount) =
                        GAMMA_DOUBLE_STAR * performanceTilde(
                        itCount) * controlStore(itCount,tCount);
164
165
166
167             end
168
169
170         end
171
172     end
173
174     %sum down columns (don't do for iteration == 1, will
            collapse to scalar)
175     if iteration > 1
176
177         u_star = sum(u_star_store);
178         u_double_star = sum(u_double_star_store);
179
180     end
181
182     % sum the control inputs for the iteration
183     controlSignal(iteration,:) = u_star + u_double_star;
184
185
186
```

```
187     %   Generate the control_ILC signal
188     if iteration > 1
189
190         control_ILC = [t' controlSignal(iteration,:)'];
191
192     end
193
194     %*****************************************************
195     %************ CHOOSE ONE MODEL *********************
196     %*****************************************************
197     %   Simulate Model
198     sim('DWILC_resistor_model')
199     sim('DWILC_thermal_model')
200     %*****************************************************
201     %*****************************************************
202
203     simulation_output = sys_out.Data;
204     outputStore(iteration,:) = simulation_output';
205     controlStore(iteration,:) = control_out.Data';
206
207     %   error signal
208     error(iteration,:) = reference_signal(:,2)'-
          simulation_output';
209
210     %   Calculate error derivative
211     end_time = t(length(t));
212     e_matrix = [t' error(iteration,:)'];
213     tau_f = .1;
214     sim('calc_deriv');
215     errorRateVec = e_dot_filt;
216     errorRate(iteration,:) = errorRateVec';
217
218     % calculate RMS error for storage
219     rmsError(iteration) = rms(error(iteration,:));
220
221
222 end
223
224 % close wait bar
225 close(wait);
```